MDP-based Motion Planning for Grasping in Dynamic Scenarios

Steffen Müller, Benedict Stephan and Horst-Michael Gross

Abstract— **Path planning for robotic manipulation is a well understood topic as long as the execution of the plan takes place in a static scene. Unfortunately, for applications involving human interaction partners a dynamic obstacle configuration has to be considered. Furthermore, if it comes to grasping objects from a human hand, there is not a single goal position and the optimal grasping configuration may change during the execution of the grasp movement. This makes a continuous re-planning in a loop necessary. Besides efficiency and security concerns, such periodic planning raises the additional requirement of consistency, which is hard to achieve with traditional sampling based planners. We present an online capable planner for continuous control of a robotic grasp task. The planner additionally is able to resolve multiple possible grasp poses and additional goal functions by applying an MDP-like optimization of future rewards. Furthermore, we present a heuristic for setting edges in a probabilistic roadmap graph that improves the connectivity and keeps edge count low.**

I. INTRODUCTION

The work presented in this paper is part of the SONARO[1] and E4SM[2] research projects, which aim at smart object handover between human and mobile robots in industrial human robot collaboration scenarios. The demands for a new controller for the robotic arm arises from the security issues when a robot manipulator tries to grasp an object held by a human hand. Due to the possible movements of the hand, we have to consider changing obstacle situations and also changing goal positions for the grasp, while the robot is executing the grasp. Existing open source software like ROS MoveIt[3] is not able to handle these dynamic tasks, causing the development of a new planning and control framework for our robots.

In the past reinforcement learning methods in the field of robotics [1] have been received more interest and proved to be very successful. In contrast to classical path planning approaches in that domain the task to achieve is not defined explicitly as a discrete goal position, but the desired behavior is defined implicitly by a scalar reward function that defines the desirability of certain states and actions the robot can reach. From this reward function an optimal policy can be derived, which defines the best action to do in each possible state. A Markov Decision Process (MDP) [2] is the fundamental model behind many reinforcement learning approaches. It defines the possible state and action space as well as the reward function. The aim of the learning methods is to identify the internal structure of the process and extract the optimal policy based on observed state transitions and received rewards. This is tedious but can be shortened, if the process model is known. Therefore, we want to benefit from the global policy and the implicit formulation of the goal by using a fully observable MDP without any incremental learning.

The perception of the environmental state is permanently changing the reward function and the process model (state action spaces) of our robot's MDP. A real-time capable solution of that MDP for the optimal policy allows to extract the immediate control commands for our robots in form of a local trajectory covering only the current planning time interval. The MDP operates on a state and actions space, which is defined by the high dimensional configuration space (C-space) [3] of the robot's joints. In order to make a real-time solution of the MDP possible, the state and action space have been discretized. Only valid, which means collision free, states and transitions are considered in order to limit the complexity. Thus, there are two processes involved. First is maintaining the set of sampled states and possible actions in form of a graph of sampled nodes in C-space and their possible transitions. This is similar to a probabilistic road map (PRM) [4]. Second process involved is the solver of the MDP which extracts the currently best policy (trajectory to follow along the graph edges). Both processes are executed in a loop in order to react in real time to changing situations.

II. RELATED WORK

Motion planning for high dimensional robot C-spaces in the last years has become efficient with the introduction of sampling based planners [5] like rapidly exploring random trees (RRT) and its derivatives. For application in a dynamically changing environment the planner in a closed loop operation has to be fast enough to find a solution within a hard deadline. Even if there are closed loop versions of RRT [6] for two dimensional navigation, for the high dimensional space RRT may be too slow when a consistent and optimal solution has to be found. A suboptimal (only collision free) solution can be found quickly, while optimality with respect to additional constraints of the path (for RRT*) is only reached with longer planning times asymptotically. There are also methods like RRT*FND [7] trying to prevent a complete reset of the found connectivity structure, when a new obstacle is detected, but another approach to reusing a plan in closed loop scenarios may be multi query planners

All Authors are with Neuroinformatics and Cognitive Robotics Lab, Technische Universität Ilmenau, 98694 Ilmenau, Germany.

This work has received funding from Free State of Thuringia from the European Social Fund (ESF) for the research group SONARO and from the Carl Zeiss Foundation as part of the project Engineering for Smart Manufacturing (E4SM) under grant agreement no. P2017-01-005

978-1-6654-1213-1/21/$31.00 ©2021 IEEE

[1]http://www.sonaro-projekt.de/

[2]https://www.e4sm-projekt.de

[3]https://moveit.ros.org/

like the PRM planners. Here a consistent graph of reachable states is computed, where a plan can be derived from different start points and also to changing goal poses as long as the obstacles do not change. In case of updated obstacles a collision check may be necessary for the existing roadmap graph, which can become expensive. In [8] a Lazy-PRM has been introduced, where collision checks are updated only along the current path. Replanning on an updated graph and collision checks on the path are done in a loop until a solution is found. The efficiency of this approach led to its integration into our system.

For the construction of a roadmap graph there are two essential questions involved determining the effectiveness and efficiency with respect to the quality of generated paths. First, where to sample the next nodes? And second, to which neighbors are they to be connected? In [9] an overview of different techniques concerning the two aspects is given.

The number of connections in the graph is essential for finding straight paths, but checking a connection for collisions is expensive, hence one tries to minimize the number of edges. In a graph with a variable density of nodes, which is desirable for increasing the accuracy in the proximity of the currently selected path and goal, the neighborhood can not be defined by a simple distance threshold. In [10] a Delaunay triangulation in the planning space is used in order to avoid redundant edges in the graph, but this is very expensive to construct especially in the high dimensional C-space. We developed a heuristic that tries to approximate the Delaunay triangulation in the edges and concentrates graph nodes at the relevant parts of the C-space.

Once a roadmap graph exists the task objective needs to be defined and an optimal path through the graph needs to be found with respect to that objective. Usually, the robot just gets one desired goal position, which is converted to C-space by means of an inverse kinematic solver [11]. Then the goal is to find the shortest path on the roadmap graph connecting the current state and that goal position.

In contrast, in the domain of stochastic control and reinforcement learning, the goal usually is not just a discrete position, but it is defined implicitly by means of a reward function rating individual actions, while the system aims at collecting as much reward as possible over time [12]. The MDP is an example for a model of that kind. This reward function approach offers the opportunity for introducing additional constraints or alternative goals rather than just minimizing the distance to a goal state, Furthermore, the inverse kinematic problem can be avoided. Therefore, we tried to merge together a PRM and an MDP solver.

In the following, the setup of the system and the definition of the objective function for grasping dynamic objects is discussed. Then the roadmap algorithm and the solver for the MDP are explained, before some analysis of the system applied to a real world setup is presented.

III. OVERVIEW OF THE MOTION PLANNER

An in the loop motion planner should be as fast, as the perception system of the robot. That means it can consider the current knowledge of the robots environment with obstacles, humans, and objects to grasp, in order to generate a movement trajectory for the individual joints of the robot's kinematic chain that covers the next time interval. Although, raw data of the environment such as depth data can be acquired by RGB-D cameras at a rate of 30Hz, the recognition of more complex features, such as human hands and objects, generally runs at a much slower rate. In our system, the slowest recognition module is the segmentation of human hands and objects, running at 5Hz on an Nvidia Jetson Xavier mobile GPU. Therefore, the motion planner also runs at 5Hz and provides trajectories for the movable joints, while it has to operate on only one CPU core of the robot's on board PC.

The most time consuming part for the motion planner is the collision test. In our system, we use a distance transform [13] of the NDT voxel map [14] of the arms' operation space. The robot's geometry is approximated as a tree of spheres, that efficiently can be tested for distances to obstacles in the distance transformed map. Also self collision tests are realized by means of that sphere tree of the robot's links.

In our application the robot has to grasp hand held objects. Therefore, the object's 6d pose is tracked and a set of suitable grasp poses for the robot's end effector is generated relative to the movable objects pose (see. Fig. 2 right). Each of these potential grasp poses is ranked by an estimated quality value q_i. Since parts of the grasp poses might be occluded by the holding hand or are not reachable for the robot otherwise, we decided to formulate an objective function that rewards the robot, if it reaches one of the possible poses (see below). Then, we let the MDP solver select implicitly among the possible grasp poses, while considering other aspects like obstacle distances and path length, rather than selecting the best grasp pose explicitly as implemented in ROS MoveIt.

A. Problem Definition

The possible states for the robot arms are represented by the nodes $\mathrm{s} \in N$ of a roadmap graph $R = (N, E)$. In each state s the possible actions a for the robot are either to stay ($a_{\mathbf{s}}$) or to take one of the adjacent edges ($a_{\mathbf{s}'}$) in E yielding to a neighboring state s'.

The individual states or nodes are more or less usefull for solving a task. Therefore, task dependent rewards $R_n(\mathrm{s})$ and $R_e(\mathrm{s}, a_{\mathbf{s}'})$ for the nodes and edges respectively are defined by a task dependent set of objective functions.

Let $O = \{(w_i, r_n^i, r_e^i)\}$ be this set of active objectives with an influence weight w_i each defining its own reward functions $r_n^i(\mathrm{s})$ and $r_e^i(\mathrm{s}, a_{\mathbf{s}'})$. The actual reward for a node s is then computed by

$$R_n(\mathrm{s}) = \sum_i w_i r_n^i(\mathrm{s}) \tag{1}$$

and the reward for a transition between s and s′ by

$$R_e(\mathrm{s}, a_{\mathbf{s}'}) = \sum_i w_i r_e^i(\mathrm{s}, \mathrm{s}') \tag{2}$$

To solve for a reward maximizing policy in an MDP-like manner, the immediate reward of an action in a state has to be defined. In contrast to ordinary discrete MDPs, the actions have different durations depending on the distance to go between two states. Thus, the overall reward for an action results from integrating the node rewards and edge rewards over the time $l_{s,s'}$ it takes to traverse an edge from $\mathbf{s}$ to $\mathbf{s}'$.

This can be done by the following equation:

$$r(\mathbf{s}, a_{\mathbf{s}'}) = \left(\frac{R_n(\mathbf{s}) + R_n(\mathbf{s}')}{2} + R_e(\mathbf{s}, a_{\mathbf{s}'}) \right) l_{s,s'} \tag{3}$$

A policy $\pi(\mathbf{s})$ is a mapping from states to the action to be performed in that state. The aim of a MDP solver is to find a policy π^* that maximizes the reward gained in the future.

$$\pi^* = \underset{\pi}{\operatorname{argmax}} \int_{t=0}^{\infty} r(\mathbf{s}_t, \pi(\mathbf{s}_t))\gamma^t \partial t \tag{4}$$

The discount factor $\gamma \in]0, 1[$ limits the time horizon. High values result in a behavior, seeking only for the state with the highest reward. On the other side, lower values take more into account how much reward the robot will gain on the way in its immediate future.

A movement trajectory $T = (\mathbf{s}_1, \ldots, \mathbf{s}_x)$ can then be queried from the policy $\pi^*(\mathbf{s})$ by starting at the current robot state and following the deterministic actions of π^* until the end of the planning interval.

In sec. V we will describe how the optimal node sequence T can be found given a set of constraints.

B. Objective Functions

For planning trajectories for our task of grasping moving objects, we defined the following objectives.

a) Obstacle distance: This objective evaluates the distance $d_{obs}(\mathbf{s})$ of a C-space configuration $\mathbf{s}$ and obstacles. The reward starts at -1 for a distance of zero and increases linearly up to 0 at a defined safety distance d_{min}. This objective causes the robot to keep a safety distance to obstacles if possible.

$$r_n^{obst}(\mathbf{s}) = min\{\tfrac{d_{obs}(\mathbf{s})}{d_{min}} - 1, 0\} \tag{5}$$

b) Distance of link to target pose: This objective is used for moving to a specific pose g with the chosen link of the kinematic chain. The reward is computed by

$$r_n^{linkdist}(\mathbf{s}) = e^{-d(f_i(\mathbf{s}),g)/\sigma} \tag{6}$$

where $\mathbf{s}$ is the C-space configuration at the graph node of interest that is converted into a pose in euclidean space via the forward model $f_i(\mathbf{s})$ of the kinematic chain for the i-th link. There can be multiple objectives of this type active at a time and the planner tries to deliberate. The distance of two 6d poses $d(a, b)$ is designed to punish rotational deviation only if euclidean distance is already low.

c) Distance to grasp poses: This objective is similar to the previous objective, but multiple target poses are handled by a single objective by taking the maximum reward over all poses to avoid superposition effects.

$$r_{grasp}(\mathbf{s}) = \max_i \{q_i e^{-d(f(\mathbf{s}),g_i)/\sigma}\} \tag{7}$$

Where g_i is the i-th grasp pose and q_i its quality. Nevertheless, there will be multiple nodes with high reward in the roadmap graph if they are close to one of the grasp poses.

d) Movement Costs: In order to prefer closer targets over more distant ones, there must be a negative reward for traveling along the edges. Here, we can chose a constant $r_e(\mathbf{s}, a_{\mathbf{s}'}) = -1$, since the time needed to travel along an edge is proportional to its length in C-space. The overall reward for passing a particular edge (see Equ. 3) in the end will be the integral over time needed to pass the edge and therefore, shorter edges are preferred.

By defining objectives in this manner it is easy to incorporate additional constraints or goals into the planning algorithm.

IV. CONSTRUCTING THE ROADMAP GRAPH

The roadmap graph is constructed and maintained in the ongoing planner cycle while aiming for the following goals:

- Keep it consistent with the environment model (collision free with NDT voxel map),
- Discover connected areas of the reachable C-space,
- Optimize the resulting path (node positions) with respect to the objective function.

As explained in the introduction, we use the roadmap graph on the one hand for exploring the connectivity structure of the collision free C-space by random sampling, and on the other hand a subset of the nodes are directly used as the trajectory points of the next control sequence sent to the robot hardware. Therefore, the density of nodes in the graph needs to be high in the regions of interest, while it can be sparse in more distant parts of the state space resulting in a very heterogeneous distribution. To support this behavior, the nodes of the graph additionally estimate the node density $\rho(\mathbf{s})$ in form of the average distance to connected nodes in the neighborhood and keep track of the nodes' age $a(\mathbf{s})$, as well as the probability of them being part of an optimal path $p(\mathbf{s})$. The probability $p(\mathbf{s})$ of belonging to a path is estimated by a simple leaky flag that is set to one, once the node $\mathbf{s}$ is part of a planned trajectory and decays over time exponentially.

Through this additional information about a node we can define operations to add or remove nodes to the roadmap graph, which are done periodically in the main planning loop (see Alg. 1).

A. Adding Nodes to the Roadmap Graph

Key element for constructing the roadmap graph is the insert node operation, where we have to decide which neighbor nodes are to be connected. A desirable goal is to achieve a Delaunay triangulation which might be optimal with respect to number of edges, but is hard to achieve in high dimensional space. The Bowyer–Watson algorithm [15] can be used for maintaining a Delaunay triangulation, but it is quite expensive to find the corresponding d-simplex structures that would be required. Thus, we introduced a heuristic that might not produce an exact Delaunay triangulation, but proved to be fast and yields reasonable edge structures. By means of a parameter δ in our approach the amount of edges in the graph can be adjusted.

When introducing new graph nodes $\mathbf{s}$, first, candidates for connecting edges have to be defined, which then are to be tested for collisions. In [16] a fixed distance threshold is analyzed, which is the usual method, but in very unevenly sampled graphs a fixed radius will fail to yield reasonable connections. Therefore, we decided to use the k nearest neighbor nodes (e.g. k=15) as a candidate set instead. This condition helps adapting the amount of connections to the local node density, but is not enough. We defined a further condition that ensures skipping redundant edges. For creating an edge to a neighbor node $\mathbf{n}$, where a node $\mathbf{m}$ exists that is connected to $\mathbf{n}$ and $\mathbf{s}$, the following condition must hold:

$$|\mathbf{s}-\mathbf{m}|+|\mathbf{m}-\mathbf{n}|>\delta|\mathbf{s}-\mathbf{n}| \tag{8}$$

That means, there must not be a one hop detour that is shorter than the direct way times a detour factor $\delta \in [1,\infty[$. This can easily be checked while traversing only the connected nodes of $\mathbf{n}$.

Only if this condition holds, we have to execute the expensive collision checks for the new edge and if it is collision free it can be inserted into the graph. Afterwards, when a triangle $[\mathbf{s},\mathbf{n},\mathbf{m}]$ has been created, we can check the condition (8) also for the existing edges and remove potentially dominated edges.

Fig. 1 shows the resulting edge structure for a small example with different node densities. In Sec. VI we will show how the number of edges created depends on δ and influences update speed and success rate of a dynamic grasping task.

B. Pruning the Roadmap Graph

To prevent the number of nodes from growing indefinitely, an operation for removing nodes is necessary. Here, the additional node properties $\rho(\mathbf{s})$, $a(\mathbf{s})$, and $p(\mathbf{s})$ will be considered for computing a probability $P(D_s)$ for deleting node $\mathbf{s}$.

$$P(D_s)=\rho(\mathbf{s})^{-\nu}\left(1-p(\mathbf{s})\right)\left(1-e^{-a(\mathbf{s})/\tau}\right) \tag{9}$$

Then, the number of nodes to be removed are drawn proportionally to $P(D_s)$. The definition of the probability tries to ensure, that necessary nodes, that are part of the optimal path, survive for keeping the plan stable. Additionally, outdated nodes are more likely to be removed, as nodes in high density areas are. The selection pressure parameter $\nu>0$ guides the influence of the local node density and time constant τ guides the influence of the age.

C. Roadmap Graph Optimization

As already described, we try to guide the node sampling in a way, that considers the current task to be executed and also keeps up the consistency with the collision model. Since it is too expensive to check all edges in the existing graph for collision in each planning cycle, we only compute obstacle distances for all the nodes, which is necessary anyway to update the reward function $r_n^{obst}(\mathbf{s})$. By removing nodes that

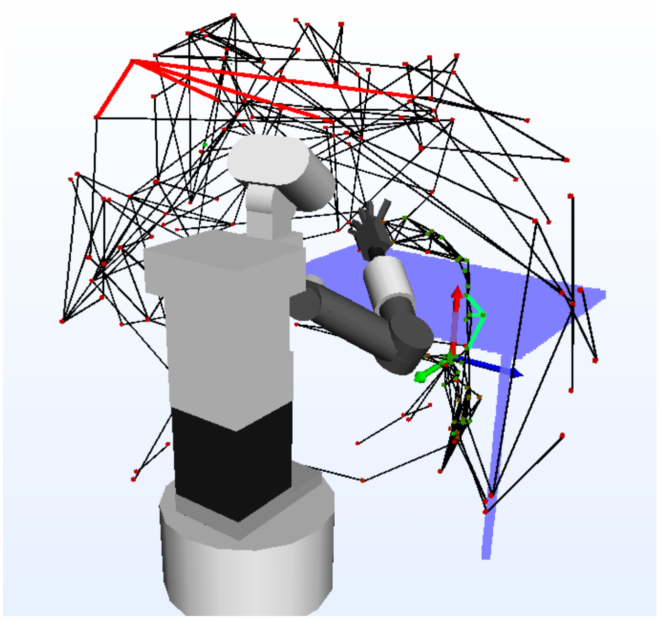

Fig. 1: Example of a roadmap graph with 200 nodes for planning a trajectory for the 7 dof arm of the TIAGo robot ($\delta=1.3$). Target is the big axes beneath the table. The color of the nodes codes for the estimated average neighbor node distance $\rho(\mathbf{n})$ which is obviously smaller along the suitable path (green nodes). The highlighted edges (red and cyan) show the adaptive connectivity in more and less dense areas respectively.

are in collision, the majority of edges leading in collision will be removed as well. Only long edges can remain, where intermediate poses lead into collision. In order to consider these as well, we establish an inner loop in the planning cycle that is repeated as long as the available planning time is not exceeded. In that loop, a path on the existing graph is planned as described in Sec. V and then all edges involved in the resulting trajectory are tested for collision and removed if necessary. Since that inner loop is repeated multiple times in a planning interval, there is a good chance to find a better collision free path before the deadline.

The process in a full planning cycle is given in Alg. 1. In the following, the individual sampling strategies involved in optimizing the roadmap graph are presented.

D. Path Interpolation and Shortcutting

Since probabilistic roadmap graphs with a manageable number of nodes are rather sparse in a high dimensional C-space, the resulting trajectories tend to be suboptimal with respect to smoothness and achievable reward. We use the result of the planned trajectory to improve the graph in the relevant region in order to get better trajectories in the next cycle. Therefore, on the one hand, interpolated poses in between nodes of the found trajectory are added to the graph, and on the other hand, direct connections of arbitrary pairs of points on the path are sampled. This checks for possible shortcuts and straightens the path gained in the next cycle.

Algorithm 1: Main Planning Cycle

```
1  trajectory $T_c = \emptyset$;
2  for all nodes s in roadmap graph do
3      $a(\mathbf{s}) := a(\mathbf{s}) + \Delta t$ ; $p(\mathbf{s}) := 0.9p(\mathbf{s})$ ;
4      remove s if in collision;
5  while planning time left do
6      reduce node number to maximum like Sec.IV-B;
7      add current state c as new node;
8      compute rewards $R_n(\mathbf{s})$ and $R_e(\mathbf{s}, \mathbf{s}')$ like
         Sec.III-A;
9      $T_c$:=solve MDP on graph like Sec.V;
10     for all $\mathbf{k} \in T_c$ do
11         $p(\mathbf{k}) := 1$
12     check for colliding edges in $T_c$ and remove them;
13     sample shortcuts on $T_c$ like Sec. IV-D;
14     sample along $T_c$ like Sec. IV-D;
15     optimize endpoint of $T_c$ like Sec. IV-E;
16     sample proportional to $R_n(\mathbf{s})$ like Sec. IV-E;
17     sample in full C-space like Sec. IV-F;
18 return $T_c$;
```

In order to explore also laterally to the path, the new points get offset by a Gaussian noise.

E. Optimizing the Trajectory End Point

Due to the formulation of the task goal in the form of an implicit reward function, we hardly can put exact goal points in the graph. For single objective end point position tasks, it is possible to use inverse kinematic solvers for generating the goal C-space point, but this is not possible for the general case. Thus, similar to numerical IK-solvers, we can do a gradient ascent. Instead of following the distance gradient in Euclidean space we follow the gradient of our reward function in order to optimize the terminal path node in the goal region. Since that gradient ascent might suffer from singularity points and local maxima of the reward function, additionally a couple of Gaussian distributed samples are added around the end point of the planned trajectory. This is similar to an evolutionary optimization over time.

The idea of exploring the position of maxima in the reward function can be extended to the whole graph rather than just for the end point of the currently reachable path. In this manner, we draw random nodes $\mathbf{s}$ from the roadmap graph proportional to their actual reward $R_n(\mathbf{s})$, for which the similar procedure of gradient ascent and Gaussian noise is applied in order to insert additional nodes in the promising regions with high reward.

F. Exploring the Connected C-Space

The last process of sampling pure random points from the C-space is necessary to explore the connectivity structure of the reachable subspace. Similar to an RRT (rapidly exploring random tree) planner [3], for the random sample $\mathbf{s}$ the nearest neighbor $\mathbf{n}$ in C-space is used in order to do a collision check along the connection line. The most distant, collision free point on the line $\overline{\mathbf{s},\mathbf{n}}$ that is closer than a maximum distance is added to the roadmap graph. Edges to other nodes fulfilling the usual conditions (Sec. IV-A) are also introduced.

V. PLANNING ON THE GRAPH

Given the roadmap graph $R = (N, E)$ with the nodes $\mathbf{s} \in N$ labeled with rewards $R_n(\mathbf{s})$ and the costs $R_e(\mathbf{s}, a_{\mathbf{s}'})$ and duration $l_{s,s'}$ for traveling along the edges $e_{\mathbf{s},\mathbf{s}'} = (\mathbf{s}, \mathbf{s}') \in E$, we now have to solve the problem of finding a reward maximizing sequence $T_\mathbf{c}$ of nodes starting at the current node $\mathbf{c}$. Therefore, we solve a Markov Decision Process (MDP) where in each state the possible actions are determined by the set of connected edges and the transition probabilities are one for ending in the successor state defined by the endpoint of the edge. The immediate rewards for the transition action then equals Equ. 3.

For solving MDPs in general there are established methods like value iteration or dynamic programming [17]. Key idea of these approaches is to find the expected value $V(\mathbf{s})$ for each state $\mathbf{s}$ by approximately solving the Bellman Equation

$$V(\mathbf{s}) = \max_{\mathbf{s}'} \{\gamma V(\mathbf{s}') + r(\mathbf{s}, a_{\mathbf{s}'})\} \qquad (10)$$

For our problem, which has some special conditions, there is a faster and exact solution, which will be presented in the following.

A. Problem Specific Properties

We can make the following assumptions about our problem:

- Considering node rewards R_n alone, the optimal path can not contain cycles because it is a potential field.
- Since rewards for traveling along an edge R_e are strictly negative, each loop in a path would reduce the overall reward. Thus, no loops are expected in the optimal path.
- Due to the discount factor γ, the overall reward would decrease, if the agent would rests in an intermediate point of the path rather than at the end point.
- Assuming that there are no loops and given the finite number n of states it follows, that there is one terminal state, which is reached after at most $n-1$ steps.

These observations induce the idea, that beginning from the end point we can plan the reward maximizing path to the start point in at most n iterations. The proposed approach is similar to the algorithm of Dijkstra [18] for planning shortest path on graphs. Instead of minimizing costs for moving along the edges, in our case we try to minimize the loss of reward to be expected. Unfortunately, the portion of reward gained in a node is not constant but depends on the time of arrival due to the discount factor γ (see Eqn. 4). Thus, we can not plan with the accumulated reward from the start point. What we can do is estimating the amount of reward that would accumulate when staying in a potential end point $\mathbf{s}$ forever.

$$\int_{t=t_a}^{\infty} \gamma^t r(\mathbf{s}, a_\mathbf{s}) \partial t = -\frac{\gamma^{t_a}}{\ln(\gamma)} R_n(\mathbf{s}) \qquad (11)$$

Here t_a is the time of arrival and if we arrive later, e.g. at t_b, the complete amount of remaining reward would scale down by a factor $\gamma^{t_b}\gamma^{-t_a}$.

Similarly, we can extend the estimation of the gainable reward by one step. Let the value $V(\mathbf{s})$ be the expected future reward for the optimal policy starting at $\mathbf{s}$, then the value $V(\mathbf{n})$ of a neighbor state $\mathbf{n}$ is at least:

$$V(\mathbf{n}) \geqq \frac{\gamma^{l_{n,s}}-\gamma^0}{\ln(\gamma)}\left(\frac{R_n(\mathbf{n})+R_n(\mathbf{s})}{2}+R_e(\mathbf{n},a_\mathbf{s})\right) + \gamma^{l_{n,s}}V(\mathbf{s}) \tag{12}$$

B. Algorithm for Solving the MDP

Using Equ. 12, we can now test for all incoming edges of a node $\mathbf{s}$, if it would be better to come over from a neighbor node $\mathbf{n}$ rather than staying there. If the result of Equ. 12 is higher than the current estimate of achievable reward in node $\mathbf{n}$, we can update the policy in $\mathbf{n}$ to follow the edge to $\mathbf{s}$ and with it the achievable reward for a trajectory starting at $\mathbf{n}$. This is the estimate of the value $V(\mathbf{n})$ of node $\mathbf{n}$.

The comparison only yields a final result, if we can know, that the estimate for node $\mathbf{s}$ will not increase later in the computation. If the node with the global maximum value is considered next, there is no possibility to increase the value of that node by coming over from a neighbor, since edge rewards are strictly negative and symmetric. The planning algorithm, therefore, pushes all nodes in a priority queue P using the expected reward for staying in the nodes forever (Eqn. 11) as the priority. Then the queue is processed starting from the highest element while testing for possible improvements of neighboring nodes. In order to find the final path, each node holds a pointer S to the most profitable successor, which is kept up to date when a better successor is found. Then the path T_c can queried by following the successor pointers until it points to itself. The whole algorithm is shown in Alg. 2.

Algorithm 2: Path Planning for Start Node $\mathbf{c}$

```
1  for all nodes s do
       /* init expected rewards                */
2      $V(\mathbf{s}) := -ln(\gamma)^{-1}R_n(\mathbf{s})$;
       /* init pointer to successor            */
3      $S(\mathbf{s}) := \mathbf{s}$;
4      push s in priority queue $P$ with prio $V(\mathbf{s})$;
5  while P not empty do
6      pop s with max $V(\mathbf{s})$ from $P$;
7      for all neighbors n of s do
8          $V^+ := \frac{\gamma^{l_{n,s}}-\gamma^0}{\ln(\gamma)}\left(\frac{R_n(\mathbf{s})+R_n(\mathbf{n})}{2}+r_e(\mathbf{n},a_\mathbf{s})\right) + \gamma^{l_{n,s}}V(\mathbf{s})$;
9          if $V^+ > V(\mathbf{n})$ then
10             $V(\mathbf{n}) := V^+$;
11             $S(\mathbf{n}) := \mathbf{s}$;
12             update $(\mathbf{n}, V(\mathbf{n}))$ in $P$;
13 $T_c$:=collect path by following $S(\mathbf{c})$;
14 return $T_c$;
```

C. Run Time Analysis

The proposed algorithm touches each node only once. Thus, it has a runtime complexity in $O(nmlog(n))$, with n being the number of nodes and m the number of edges per node. The factor $O(log(n))$ results from the priority queue, which has to be filled and updated when the planning proceeds. The practical evaluation has shown, that by means of the proposed method for maintaining the roadmap graph the number of edges can be kept within a reasonable limit that is just dependent on the number of dimensions of the C-space, the number of candidate neighbors k, and the detour factor δ. Thus, m can be seen as constant w.r.t. n which yields $O(nlog(n))$ as the overall complexity. Maintaining the roadmap graph structure is in $O(nm^2)$ if a naïve nearest neighbor search is used and the check for dominated edges has to follow m^2 neighbors.

VI. EXPERIMENTAL ANALYSIS

The proposed system has been implemented and tested on two different robot platforms. First is a TIAGo robot by Pal Robotics with a 7 dof robotic arm and a prismatic joint for raising and lowering the body (see Fig. 2) and the second platform is a Scitos X3 by Metralabs equipped with two Kinova Jaco arms each having 6 dof (visible in Fig.3).

We conducted a series of comparative experiments using the proposed motion planner in the context of grasping for a moving object. The setup for reasons of repeatability was using a simulated target object, which has been moved along a rectangular path as depicted in Fig. 2. A table served as a static obstacle causing the robot to plan longer evasive movements when the object passes behind the legs of the table or when it switches over to the other side of the table top.

As the target for the grasping action a set of 100 grasp poses (see right side of Fig. 2) had been generated using an analytic grasp pose planner each having a quality weight between 0.5 and 1. These grasp poses were attached to the moving object and have been fixed for all experiments.

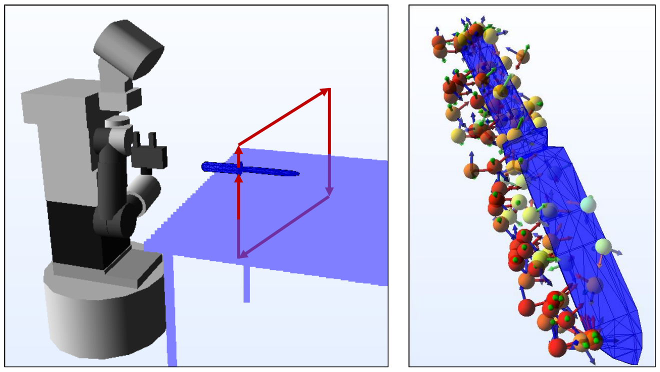

Fig. 2: Robot in the start pose for the experimental grasping setup. The target object moves along the given trajectory above and beneath the table. On the right there is the object (kitchen knife) with the set of suitable grasp poses. Color intensity is coding for the quality of the grasps.

Method	Success	Timeout	Avr. Duration	Avr. Obstacle Dist.
MDP OD	70%	30%	14.3 sec	45 mm
MDP	78%	22%	13.8 sec	37 mm
Dijkstra	59%	41%	15.1 sec	64 mm

TABLE I: Results of 1000 attempts to grasp the moving object with the proposed method (MDP OD) including obstacle distance objective and without (MDP) vs. a pure distance based approach (Dijkstra).

A grasp attempt counts as successful, if in the moment the gripper closes a grasp quality evaluation of the current gripper pose yields a valid grasp quality in the range of 0.5 to 1. If the robot did not attempt to close the gripper within 30 sec. the trial is aborted and counted as a timeout. For each trial, the robot pose is reset to its home pose and the roadmap graph is cleared. The objects start position is changed in a deterministic sequence in order to have consistent conditions for all experiments.

A. Evaluation of the MDP Solver

In a first experiment, we compare the MDP solver for the dynamic multi target task ($\gamma = 0.99$) with a simple Dijkstra algorithm, which aims to reach the roadmap graph node with the highest reward $R_n(\mathbf{s})$ on the shortest path (w.r.t. $R_e(\mathbf{s}, a_{\mathbf{s}'})$) in each planning cycle.

Table I shows that the success rate of the proposed method (MDP) with and without the objective function for obstacle distance (Sec. III-B.0.a) outperforms the naive approach (Dijkstra). The Dijkstra algorithm due to the moving object yields a very unstable goal pose, since the node with highest reward changes regularly. This causes more complicated detouring trajectories in order to reach the new best grasp pose. Therefore, the average time to grasp is higher and also the distance to obstacles along the path is higher, since it spends less time close to the target object. For the MDP solver, the success rate is much higher due to more consistent trajectory end points. The MDP accepts a suboptimal end point if the smaller costs for the path compensate the loss of reward due to the smaller grasp quality.

The experiment also shows, that the objective function considering the distance of roadmap graph configurations to obstacles has an effect, which for the Dijkstra algorithm can not be the case, since this is only considering the length of the resulting path. If the obstacle distance objective is active, the average distance to obstacles increases, which might improve safety, but on the other hand, the robot has more difficulties to fit the object between the grippers, since in the reward function the object repels the gripper. This leads to the slightly longer duration to grasp and thus a worse success rate due to more timeouts.

B. Evaluation of the Triangulation Strategy

In order to demonstrate the benefit of the proposed method for connecting nodes in the roadmap graph (Sec. IV-A), a second experiment evaluated the success rate of the dynamic grasp task in dependence of the detour factor δ. Tab. II shows that the number of edges in the graph and thus

δ	k	Succ. Rate	Time-out	Avr. # Edges	Fan Out Avr.	Max. # Edges	Fan Out Variance
7.0	15	78%	22%	3531	3.55	4168	7.5
5.0	15	72%	28%	3458	3.53	4189	7
3.0	15	**81%**	**19%**	3543	3.59	4243	7
2.0	15	80%	20%	4101	4.16	5486	6
1.7	15	76%	24%	5346	5.35	7336	8
1.5	15	69%	31%	6763	6.92	9260	14
1.3	15	73%	27%	8397	8.67	12074	25
1.1	15	76%	24%	9338	9.58	15836	33
1.0	15	74%	26%	9855	10.39	18378	51
1.0	4	70%	30%	3650	3.70	5899	4.18
2.0	4	66%	34%	2498	2.50	3429	1.90

TABLE II: Results of 100 attempts to grasp the moving object with MDP planner depending on detour factor δ. Max number of graph nodes was 2000.

also the average fan out of the nodes decreases when δ increases. Interestingly, also the success rate of the grasps has a maximum at lower connectivity in the graph (at $\delta = 3.0$).

This may result from an increased number of internal optimization cycles (Alg. 1 line 5), which is possible because the time for solving the MDP decreases on a less complex graph in the same amount of planning time.

In order to verify that the proposed method also creates an improved connection structure compared to a simple reduction of neighbor candidates, the experiments in the last rows of Tab. II have been done. It turns out, that neighbor count $k = 4$ produces also about 3600 edges in the graph, but the success rate drastically dropped, which is an indicator for a worse graph structure, since all other parameters were unchanged.

C. Real World Experiment

Since all previous experiments analysed the properties of the proposed motion planner in simulation, a last experiment is to demonstrate the capabilities in real world. For the real world problem the robot had to follow moving targets with one or both of it's Kinova Joco 6-axis arms simultaneously. The targets switch over behind obstacles periodically forcing the arms to do evasive movements. For the two armed case the two target positions were moved at different speeds. Parameters of the planner were kept the same for both experiments. Fig. 3 shows the setup whereas results are shown in Fig. 4.

For the single armed scenario the arm can follow the target until it passes the obstacle. As soon as the goal is reachable again the arm moves around the obstacle, resulting in an increase in distance. After catching up the goal can be followed closely again. For moving both arms simultaneously the results are similar to the one armed case. Once the first arm has to move around an obstacle an effect on the second arm can be observed. This is because both arm movements are represented by the same graph. Thus, if both arms are in close proximity to the moving target, the graph node density in the goal region is high enough to allow both arms to find a solution quickly. If one arm is occluded, the path has to explore new regions of the 12 dimensional C-space, where node density is low. The usable nodes in the graph are not

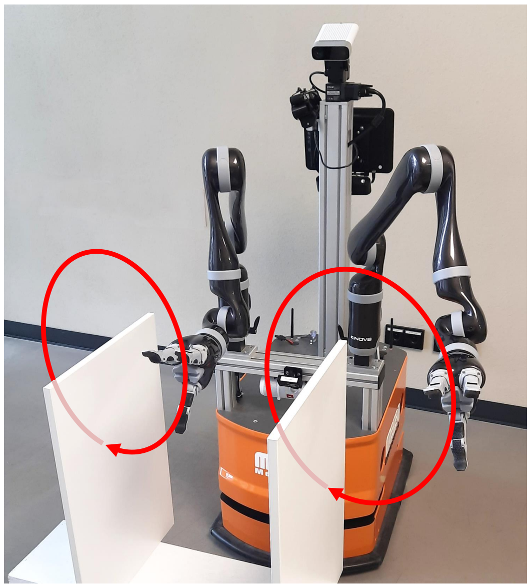

Fig. 3: Real world setup with two armed Scitos X3, paths of the goal positions and obstacles.

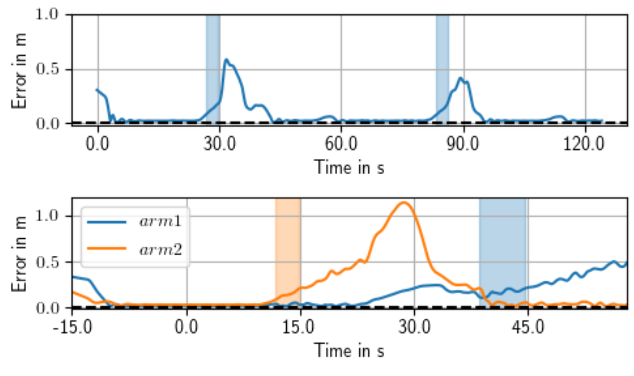

Fig. 4: Resulting distance between end effector and goal over time. Interval at which a goal position was unreachable is marked. Top shows results of single arm experiment and bottom with two arms.

optimized in position yet, forcing the second arm on a sub optimal path.

VII. CONCLUSIONS

In this paper a motion planner is proposed, working with a dynamic and local update of a probabilistic roadmap in closed loop operation. We showed the effectiveness of the mechanism for managing the connectivity in the graph in order to handle heterogeneous density of nodes and keep the number of edges low. Further, we could demonstrate, that a grasp task can be solved more effective, if the set of possible grasp poses is also considered by an implicit formulation of the goal in form of a reward for a MDP, rather than just moving to the best rated single grasp goal position on a shortest path. Additionally, we showed, that a decomposition of the reward function together with the constraints given in our motion planning scenario allow to find an exact solution for the MDP in $O(nlog(n))$, which is done in each planning cycle.

In future work grasping for real objects presented by a human will be evaluated in more detail. Unfortunately, the robust and stable tracking of handheld objects,which is a prerequisite for the motion planning, is a hard task on its own and has to be solved first.

REFERENCES

[1] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[2] E. A. Feinberg and A. Shwartz, *Handbook of Markov decision processes: methods and applications*. Springer Science & Business Media, 2012, vol. 40.

[3] B. Siciliano and O. Khatib, *Springer handbook of robotics*. springer, 2016.

[4] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[5] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[6] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli, and J. P. How, "Real-time motion planning with applications to autonomous urban driving," *IEEE Transactions on control systems technology*, vol. 17, no. 5, pp. 1105–1118, 2009.

[7] O. Adiyatov and H. A. Varol, "A novel rrt*-based algorithm for motion planning in dynamic environments," in *2017 IEEE International Conference on Mechatronics and Automation (ICMA)*. IEEE, 2017, pp. 1416–1421.

[8] H. Akbaripour and E. Masehian, "Semi-lazy probabilistic roadmap: a parameter-tuned, resilient and robust path planning method for manipulator robots," *The International Journal of Advanced Manufacturing Technology*, vol. 89, no. 5-8, pp. 1401–1430, 2017.

[9] R. Geraerts and M. H. Overmars, "A comparative study of probabilistic roadmap planners," in *Algorithmic foundations of robotics V*. Springer, 2004, pp. 43–57.

[10] P. Bhattacharya and M. L. Gavrilova, "Roadmap-based path planning-using the voronoi diagram for a clearance-based shortest path," *IEEE Robotics & Automation Magazine*, vol. 15, no. 2, pp. 58–66, 2008.

[11] A. Aristidou, J. Lasenby, Y. Chrysanthou, and A. Shamir, "Inverse kinematics techniques in computer graphics: A survey," in *Computer Graphics Forum*, vol. 37, no. 6. Wiley Online Library, 2018, pp. 35–58.

[12] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[13] I. Ragnemalm, "The euclidean distance transform in arbitrary dimensions," *Pattern Recognition Letters*, vol. 14, no. 11, pp. 883–888, 1993.

[14] E. Einhorn and H.-M. Gross, "Generic 2D/3D SLAM with NDT maps for lifelong application," in *Europ. Conf. on Mobile Robots (ECMR)*, 2013.

[15] D. F. Watson, "Computing the n-dimensional delaunay tessellation with application to voronoi polytopes," *The computer journal*, vol. 24, no. 2, pp. 167–172, 1981.

[16] K. Solovey and M. Kleinbort, "The critical radius in sampling-based motion planning," *The International Journal of Robotics Research*, vol. 39, no. 2-3, pp. 266–285, 2020.

[17] A. Kolobov, "Planning with markov decision processes: An ai perspective," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 6, no. 1, pp. 1–210, 2012.

[18] E. W. Dijkstra *et al.*, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.